A Parallel Platform for Mining General Co-Movement Patterns over Large-scale Trajectories

ABSTRACT

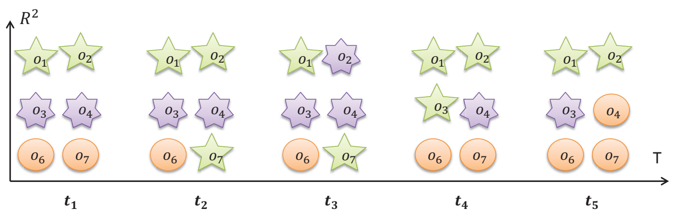
1. INTRODUCTION

The prevalence of positioning devices has drastically boosted the scale of trajectory data collection. Instances include telemetry chips for animal herding histories, GPS devices for urban transportations and wearable devices for personal moving activities. These positional data are often time-stamped and can be viewed as trajectories. Data analysis on large-scale trajectory data can benefit a wide range of applications and services, including traffic planning, animal analysis, and social recommendations, just to name a few. CITATIONS FOR THE ABOVE APPLICATIONS.

An important task of data analysis on trajectories is to discover co-moving objects. Intuitively, a *co-movement* pattern [1] consists of a set of objects that travel together for some duration. To formally define the pattern, the temporal dimension of trajectories is descritized into snapshots, where each snapshot contains the geographical information of all moving objects. Given a member size constraint n, a temporal constraint k, a *co-movement* pattern finds a cluster of objects with at least size n and close in spatial proximity for at least k snapshots. Recently, there have been several works extending the basic pattern to incorporate more advanced temporal constraints. For instance, Jeung et al. proposed *convoy* pattern, which requires the snapshots to be consecutive; Li et al. proposed *swarm* pattern, which relaxes the consecutiveness of snapshots and Li et al. proposed *platoon* pattern which imposed a *minimum local length* on the snapshots. PLEASE BE MORE SPECIFIC FOR THE THREE PATTERNS. ALSO, ADD THE CITATIONS.

Figure 1 summarizes the differences among various co-movement patterns with an illustrative example. The temporal dimension is sliced into 5 snapshots. By setting $n = 2$, the spatial clusters with at least 2 objects are displayed with the same shape. Since *Convoy* pattern requires the set of objects to be clustered for k *consecutive* snapshots, there results in only one such pattern ($\{o_3, o_4\}\{t_1, t_2, t_3\}$) when k is set to 3. *Swarm* pattern relaxes the consecutiveness of duration, thus there are three patterns discovered. *Platoon* pattern requires that each local consecutive duration needs to have at least certain length, which is indicated by an additional parameter l. When l equals 2, there are two patterns discovered. Note that $\{o_6, o_7\}\{t_1, t_4, t_5\}$ is not included in the platoon pattern since t_1 is the local consecutive snapshot with duration 1, which is less than l. CAN WE SUPPORT THE GROUP AND FLOCK PATTERN? IF YES, WHY NOT PUT THEM IN THE FIGURE?

FOR THE FIGURE: 1) ENLARGE THE FONT SIZE IN THE FIGURE 2) WHY THERE IS NO o_5. 3) WHY OBJECTS IN DIFFERENT COLORS AND SHAPES ARE NOT CLEAR. YOU CAN REMOVE COLOR. 4) PUT THE OBJECTS BELONGING TO THE SAME CLUSTER IN THE SAME LINE.


R^2
T
o_1 o_2 o_3 o_4 o_6 o_7
t_1 t_2 t_3 t_4 t_5

Convoy (n=2,k=3)	$\{o_3, o_4\}\{t_1, t_2, t_3\}$
Swarm (n=2,k=3)	$\{o_1, o_2\}\{t_1, t_2, t_4, t_5\}$, $\{o_6, o_7\}\{t_1, t_4, t_5\}$, $\{o_3, o_4\}\{t_1, t_2, t_3\}$
Platoon (n=2,k=3,l=2)	$\{o_1, o_2\}\{t_1, t_2, t_4, t_5\}$, $\{o_3, o_4\}\{t_1, t_2, t_3\}$

Figure 1: Co-moving patterns in related works

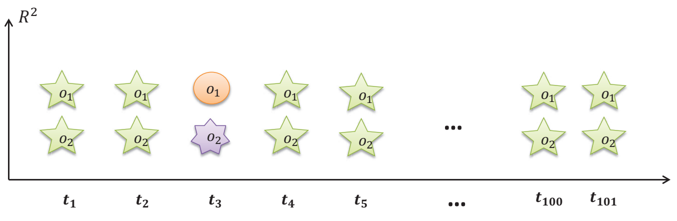


Figure 2: Miss-pattern in platoon

2. RELATED WORKS

In this section, we present a comprehensive literature review on the mining of various trajectory patterns.

FOR THE CO-MOVEMENT PATTERN, YOU NEED TO EMPHASIZE TWO THINGS: 1) THE DIFFERENCE BETWEEN PATTERNS. NEED TO CLARIFY THE PARAMETERS IN EACH MODEL. 2) CLEARLY STATE THE MINING TECHNIQUES.

2.1 Co-movement Pattern

The work most related to ours is those on *co-movement* patterns. We summarize the typical patterns as follows:

2.1.1 group

Wang et al. defined *group pattern* [10], which aims to find the set of objects travelling together at certain time intervals. In *group pattern*, groups at each snapshot is identified by a disc-based clustering method, where each cluster forms a circle within a radius. It is argued in later works [11, 12] that such disc-based clustering is not effective as *density*-based clustering where the later one may find clusters of arbitrary shapes.

2.1.2 flock

Gudmunsson et al. proposed *flock* pattern in [9, 13] and Vieria et al. followed up with an online version in [14]. A *flock* pattern tries to find the set of objects that stay in a circular ranged cluster for a minimum duration. Such a pattern is useful in detecting the moving companions. However, similar as *group pattern*, it uses disc-based clustering, which suffers the same deficiency in discovering arbitrary shaped clusters. *Flock* pattern has many derivatives. In [15], Benkert et al. studied *meet* pattern, which require the clusters in the pattern to be geographically stationary. Giannotti et al. studied *leadership* pattern [16] which requires a leader object exists for each flock cluster.

2.1.3 convoy

Jeung et al. proposed *convoy* pattern that extends *flock* pattern by replacing the disc-based clustering with *density*-based clustering. Such an relaxation brings a high complexity of repeatedly running DBSCAN [17] at every snapshot. To reduced the complexity, Jeung et al. designed a filter-refine approach which first uses simplification technique [18] to filter far away objects, and then uses coherent moving method [3] to find the exact convoy patterns. Along with *convoy* pattern, Aung et al. proposed *dynamic convoy* and *evolving convoy* patterns. In *dynamic convoy*, the cluster members are allowed to be absent briefly during the convoy lifetime, while *evolving convoy* allows the convoy to grow or shrink in cardinality during the life time. Tang et al. also addresses the online extension in [19].

2.1.4 swarm

The major argument on *convoy* pattern is that *convoy* requires the consecutiveness in the lifetime, which may lose many interesting patterns. To remedy, Li et al. proposed the *swarm* pattern [12] which completely relaxes the consecutiveness in *convoy*. In *swarm*, objects can collectively leave the cluster for a long time and then join back in later time. The only requirement in *swarm* is that each member in the cluster needs to accumulate to a certain duration. In [12], the authors proposed a depth-first search based pruning algorithm to efficiently discover *swarm* patterns.

2.1.5 platoon

Recently Li et al. argued that *swarm* is to loose in the temporal consecutiveness and proposed *platoon* pattern in [2]. In *platoon* pattern, the clusters should lasts for at least a certain during before dismiss. Meanwhile, *platoon* allow the clusters to form again at future times. Li et al. demonstrated the such extension is more general and can support swarm and convoy patterns by setting appropriate parameters. Li et al. also provide a similar depth-first search approach as in [12]. In addition, they adapted a prefix pruning method to further improve efficiency. It is notable that in both [12] and [2], authors consider the input to be the clusters at each snapshot, which ignores the clustering time.

2.2 Other Related Trajectory Patterns

Besides co-movement patterns, there are a number of other types of trajectory patterns proposed in previous works.

Kalnis et al. proposed *moving clusters* pattern [3]. In such a pattern, objects form clusters at each snapshot. For consecutive snapshots, the clusters in the pattern should have a Jaccard index greater than a threshold. Under such a scheme, the difference between cluster members in snapshots accumulates, therefore the clusters at later snapshot may be very different from those in previous snapshots. The online extension is studied by Li et al. [7]. IT'S DIFFERENCE WITH CO-MOVEMENT PATTERN IS NOT CLEAR. WHAT IS JACCARD INDEX?

In [4], Li et al. studied the *periodic* pattern, which mines objects with periodic behaviors. It is commented in [8] that *periodic* pattern is unsuitable for discovering movements, since it is unreasonable to expect an object to repeat its behavior exactly during each time period considered. AGAIN, WHAT'S YOUR POINT? YOU MEAN SUCH A WORK IS MEANINGLESS? THEN, WHY BOTHER TO MENTION IT HERE?

Zhang et al. proposed the *gathering* pattern in [5]. It is similar to *flock* pattern [9] but with the relaxation on the members of clusters. Instead of fixing the members in clusters as in [9], *gathering* pattern allows members in clusters leave and join during the pattern duration. Since it relaxes the member constraints, it is unable to model co-movement patterns. HOW TO DEFINE CO-MOVEMENT PATTERN? IS THERE A PREVIOUS "FORMAL" DEFINITION? OR IS DEFINED BY YOU? THIS ONE LOOKS LIKE OBJECTS CO-MOVE. WHY IT IS NOT A CO-MOVEMENT PATTERN?

Jinno et al. recently studied the problem of processing T- pattern [8] in parallel platform [6]. A T-pattern discovers a set of objects visiting the same the place in a close time interval. Such a pattern differs from moving object pattern in that T-pattern does not consider the movement of objects. Jinno et al. in [6] designed a MapReduce based algorithm for efficiently support T-pattern discovery. However, as the nature of differences between the patterns, their work cannot directly applied on the co-moving object pattern discovery. Li et al. recently proposed a framework of processing online *evolving group* pattern [7]. The *evolving group* is similar to *moving cluster* pattern with focus on the member updates in clusters, which is different with *co-movement* pattern. Moreover the framework is developed for centralized system, thus is different with our work.

2.3 Trajectory Clustering

Another related field is trajectory clustering [20, 21, 22]. Lee et al. proposed a partition and group algorithm in [21] TO SOLVE WHAT PROBLEM?. Their clustering method does not consider the temporal constraint and groups trajectories from different time points together. WHY EMPHAISIS THIS? OUR CLUSTERING IS ALSO CONDUCTED IN EACH SNAPSHOT.

Li et al. proposed a *micro-clustering* technique [22] to cluster moving objects based on their moving directions. However, its distance measured is defined upon the entire trajectory and cannot be applied in our problem to mine local patterns.

In [20], He et al. deployed an implementation of DBSCAN on MapReduce. They decouple the dependency of original DBSCAN algorithm into a four-stage parallel process. However their method only focuses on DBSCAN for one datasets, where exploiting the relationship between multiple DBSCANs remains unexplored. I DON'T UNDERSTAND WHAT YOU MEAN. ONE DATASET? MULTIPLE DBSCANS?

3. DEFINITIONS

Let $\mathbb{O} = \{o_1, o_2, ..., o_n\}$ denote the set of objects in concern, $\mathbb{T} = \{1, 2, ..., m\}$ be the descritized sequence of timestamps. We use $T[i]$ to denote the i^{th} entry in a time sequence.

Given a time sequence $T \subseteq \mathbb{T}$, let $|T|$ be the cardinality of the sequence. We say a sequence T is *(fully)consecutive* if and only if $\forall T[i] \in T, T[i+1] \in T$, $T[i+1] = T[i] + 1$. A subsequence T^m is *maximally consecutive* [2] wrt. T if and only if $T^m \subseteq T$ and $\nexists T^{m'}, T^m \subseteq T^{m'} \wedge T^{m'}$ is consecutive. For example, let $T = \{1, 2, 3, 5, 6\}$, then $T_1 = \{1, 2, 3\}$ and $T_2 = \{5, 6\}$ are two maximally consecutive subsequences wrt. T. We then define the L-consecutiveness as follows:

Definition 1 (L-consecutive)**.** *A time sequence T is L-consecutive iff $\forall$ its maximal consecutive subsequences T^m, $|T^m| \geq L$.*

Take $T = \{1, 2, 3, 5, 6\}$ as an example. T is not 3-consecutive since $\{5, 6\}$ is a maximally consecutive subsequence with cardinality less than 3. T is 2-consecutive since both its two maximally consecutive subsequences are of the sizes greater or equal to 2. It is notable that, for a sequence T, all its maximally consecutive subsequences are non-overlapping.

We then define the G-separateness as follows:

Definition 2 (G-separated)**.** *A time sequence T is G-separated iff $\forall T[i], T[i+1] \in T, T[i+1] - T[i] \leq G$*

Take $T = 1, 2, 3, 5, 6$ as an example. T is 2-separated since the gap between its maximally consecutive subsequnces is $5 - 3 = 2$. However, T is not 1-separated. Indeed, 1-separated infers that T is fully consecutive.

Given a time sequence t, objects locations at t collectively forms a *snapshot*[1].Objects in a snapshot can thus be clustered based on the closeness of their locations. Let $C_t(o_i)$ be the cluster which o_i belongs to at time t, we can then define the generalized co-moving pattern as follows:

Definition 3 (Generalized Co-Movement Pattern)**.** *A generalized co-movement pattern $GCMP(M, K, L, G) = \langle O, T\rangle$ is a pair containing an object set $O \subseteq \mathbb{O}$ and a sequence $T \subseteq \mathbb{T}$, with the following constraints:*

1. *Closeness: $\forall o_i, o_j \in O, \forall t \in T, C_t(o_i) = C_t(o_j)$*
2. *Significance: $|O| \geq M$*
3. *Duration: $|T| \geq K$*
4. *Consecutiveness: T is L-consecutive*
5. *Separateness: T is G-separated*

[1]Missing timestamps can be interpreted using linear interpolation [11]

Tr_i	Trajectory of object i
S_t	Snapshot of objects at time t
$\mathbb{O}$	Set of objects
$\mathbb{T}$	Global time sequence
$C_t(o)$	the cluster of object o at time t
Sr_i	The star structure of object i

Table 1: Notions that will be used

The generalized co-movement pattern retains the patterns that discovered by existing techniques (convoy, swarm and platoon). In particular, by setting $G = |T|$, we achieve the *platoon* pattern. By setting $G = |T|, L = 1$, we achieve the *swarm* pattern. Finally by setting $G = 1$, we achieve the *convoy* pattern. In addition to covering existing patterns, the generalized co-movement pattern avoids the *miss-pattern* problem in *platoon* pattern. As suggested in Figure 2, $\{100, 101\}$ will be included in the platoon pattern, however since they're too far away in temporal domain, this pattern is not prominent. By setting appropriate G, we are able to prune these two snapshots. Notice that such a new pattern cannot be modeled by existing patterns.

It is notable that the number of patterns in GCMP is exponential. To control the number of output patterns, we noticed that, for two pattern result P_1, P_2, if $P_1.O \subseteq P_2.O$ and P_2 is a proper pattern, then P_1 is also a proper pattern. Based on this observation, we define the *closed generalized co-movement pattern* as follows:

Definition 4 (Closed Generalized Co-moving Pattern)**.** *A generalized co-moving pattern $P = \langle O, T\rangle$ is a closed generalized co-moving pattern if and only if there does not exists another generalized co-moving pattern P' s.t. $P'.O \supseteq P.O$.*

For example, let $n = 2, k = 2, l = 1, g = 1$, the pattern $\{o_1, o_2\}\{1, 2, 3, 4\}$ is not a closed pattern, while $\{o_1, o_2, o_3\}$ $\{1, 2, 3, 4\}$ is a closed pattern.The closed pattern avoids to output duplicate information, thus making the result patterns more compact.

Although the generalized co-moving pattern is free from the clustering methods used at each snapshot, as suggested in [11], the *density*-based clustering method is better in detecting objects with arbitrary spatio-shaped clusters. Therefore in this paper, we mainly consider density-based clustering.

The goal of this paper is to present a parallel solution for discovering GCMP from large-scale trajectory data.

Before we move on to the algorithmic part, we list the notations that are used in the following sections.

4. PRELIMINARY ON APACHE SPARK

Apache Spark is an open sourced modern parallel processing platform based on Resilient-Distributed-Dataset (RDD). Each RDD is able to take an action (including Map, Reduce, GroupByKey etc.) and then transforms to a new RDD. Algorithms in Spark is implemented by supplying various actions to an input RDD. Each RDD in Spark has a predefined and fixed number of partitions, where each partition forms a task assigned to an executor.

Compared to MapReduce, Spark utilizes distributed memory to cache the programming data modeled as RDDs, which brings computational benefits [23]. In order to fully utilize the memory, Spark creates one thread for each task and then multiple tasks belonging to the same JVM share the available memory. Due to threading, there is a paradigm shift from MapReduce to Spark. In MapReduce, the cost of starting a task is expensive, as each task in MapReduce requires a dedicated JVM process. While in Spark, the cost of starting a task is negligible since each task is done by a thread. Therefore in Spark, we may create many more parallel tasks than in MapReduce before reaching to the system limitation.

5. MINING GENERALIZED CO-MOVING PATTERN

There are two stages in mining GCMP. The first stage is to cluster objects in each snapshots. Then the seconds stage is to mine patterns from clusters in parallel. The overview of the parallel approach is shown in Figure 3. As shown, trajectory data is initially stored in HDFS. The first stage is to read trajectories and cluster objects in the same snapshots. Since the objects at each snapshots are independent, this stage can be easily performed in parallel. The first stage involves network IO. Afterwards, the clusters in each snapshots are shuffled and replicated to make various partitions. In order to discover all patterns, we require a partitioning method to be *complete* and *Sound*.

Definition 5 (Completeness and Soundness)**.** *Let a partitioning method $\mathbb{P}$ partitions original trajectories TR into multiple parts, $Par_1, ..., Par_m$. $\mathbb{P}$ is complete if for every pattern P that is valid in TR, $\exists Par_i$ such that P is valid in Par_i. $\mathbb{P}$ is sound if for all patterns P that is valid in any Par_i, it is also valid in TR.*

The completeness ensures that no real patterns are missed out. The soundness ensures that no false patterns are reported. If a partition method is both sound and complete, we are then able to mining patterns in each resulted partition in parallel.

Depends on the pattern mining strategy, the size of shuffle and replication may differ. We show that if the pattern mining strategy is not selected carefully, the second stage will suffer from large data replication and shuffling.

5.1 Temporal Replication and Mining

The straightforward strategy of parallelizing the second stage is to vertically partition the trajectories based on snapshots. A simple but effective method is to group neighborhood snapshots into a partition, such that every possible pattern can be mined within some of the partitions. In order to achieve the *completeness*, some of the snapshots need to be replicated on multiple executors. We call this method

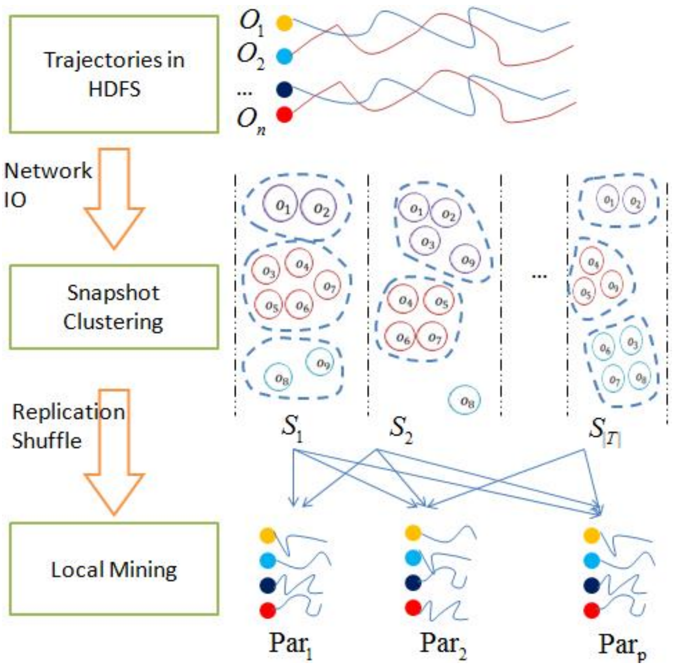


Figure 3: System Flow of Mining GCMP

the *Temporal Replication and Mining* approach. The algorithm is presented as in Algorithm 1.

Algorithm 1 Temporal Replication and Mining

```
Require: list of $\langle t, S_t \rangle$ pairs
  Map Phase
  for all $\langle t, S_t \rangle$ do
      for all $i \in 1...(K-1)*G+K$ do
          emit a $\langle t-i, S_t \rangle$ pair
      end for
  end for
  Shuffle Phase
  for all $\langle t, S \rangle$ pair do
      group-by $t$, emit a $\langle t, Par_t \rangle$,
      where $Par_t = \{S_t, S_{t+1}, ..S_{t+(K-1)*G+K}\}$
  end for
  Reduce Phase
  for all $\langle t, Par_t \rangle$ do
      minePattern($Par_t$)
  end for
```

5.1.1 Temporal Replication Partition

In order to reduce the shuffling cost, the data to be replicated should be kept to a minimum. However, as suggested in the following theorem, the minimum replication of a snapshot is not small:

Theorem 1 (Soundness and Completeness of Replication)**.** *For each snapshot S_t, a partition $Par_t = \{S_t, ..., S_{t+(\lfloor \frac{K}{L} - 1 \rfloor * G+K)}\}$ is created. Such a partition method is sound and complete.*

Proof. The soundness of partition is trivially true by definition. Given a valid pattern P, let $T' \subseteq P.T$ be a subsequence of $P.T$ which conforms to K, L, G. Note that there could be many qualified T' of T. Then, let the i^{th} local-consecutive part of T' be l_i and let the i^{th} gap of T' be g_i. Then, the size of T' can be written as $\Sigma_i(l_i + g_i)$. Since T' conforms to K, L, G, then $\Sigma_i(l_i) \geq K$, and $K \geq l_i \geq L$, $g_i \leq G$. Therefore, among all possible T's, the minimum size is $\lfloor \frac{K}{L} - 1 \rfloor * G + K$. Thus ensuring each Par_t to be of that size would capture one of the T's, therefore the pattern P is valid in Par_t. This proves the completeness of the partitioning method. □

Utilizing Theorem 1, we create, for each snapshot S_t, a partition containing its next $\lfloor \frac{K}{L} - 1 \rfloor * G + K$ snapshots. Each partition is then sent to the executors as a task. Since any global pattern must exists in one of the partitions, we can mine the patterns from each partitions independently, without loss of patterns.

5.1.2 *Temporal Replication Mining*

After replication, each task in Spark will process an partition Par_i. The next step is to mine the GCMPs from Par_i. We notice that, for each partition, we only need to find the patterns that are contained in the first snapshots. Therefore we design a line-sweep method for mining such patterns, which is the variant of the Coherent-Moving-Clustering method in [11].

The Temporal Replication and Mining approach though achieves parallelism from independent partitions, it requires to replicate the data multiple times. Specifically, each snapshots are copied $(K-1) * G + K$ times. In swarm case, G is as large as $|\mathbb{T}|$. In such a case, it is equivalent to replicate the entire dataset to each executor, which is clearly inefficient.

5.2 Star Partition and Mining

To develop a method that achieves parallelism under any pattern parameters, we propose the *Star Partition and Mining* (SPM) method. In SPM, we design a novel object-based partition method named star partition. A start partition partitions trajectories in the object domain rather than temporal domain. After partitioning, we design the *Apriori*-like method to mine the patterns out of each partition independently. The overview of the star partition and mining is as in Algorithm 2.

Algorithm 2 Star Partition and Mining

```
Require: list of ⟨$t, S_t$⟩ pairs
  Map phase
  for all $C \in S_t$ do
      for all $(o_1, o_2) \in C \times C$ do
          emit a $\langle o_1, o_2, \{t\} \rangle$ triplet
      end for
  end for
  Shuffle phase
  for all $\langle o_1, o_2, \{t\} \rangle$ triplets do
      group-by $o_1$, emit $\langle o_1, Sr_{o_1} \rangle$
      group-by $o_2$, emit $\langle o_2, Sr_{o_2} \rangle$
  end for
  Reduce phase
  for all $\langle o, Sr_o \rangle$ do
      Apriori($Sr_o$)
  end for
```

5.2.1 *Star Partition*

The purpose of star partition is to find the group of objects that could potentially form a pattern. In order to achieve parallelism, we group, for each object o, all other objects that connect to o in some snapshots. By so doing, the patterns containing o can be discovered within each groups. Technically, we create a connection graph to represent the connectivity among objects. A connection graph is an undirected graph $G = (V : E)$, where each $v \in V$ represents an object. An edge $e(s,t) = ET$ contains all the timestamps where s, t are in the same cluster, i.e., $\forall t \in ET, C_t(s) = C_t(t)$. Given a vertex s, the star of s, Sr_s is the set of incidental edges of s in G. An example of star partition is shown in Figure 4.

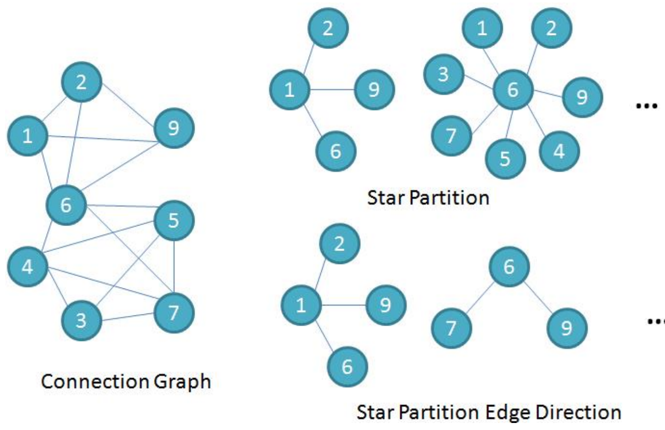


Figure 4: Example of Star Partition

In star partition, trajectories are partitioned based on objects and their stars. Then each partition is send to an executor as a task. Indeed, a star Sr_s can be viewed as a subset of original trajectories. This is done by treating each vertex in Sr_s as an object. For s, the time sequence is the union of all edges in Sr_s. For vertex $v \neq s$, the time sequence is the edge (s, v). Then, we state the completeness of star-partition as in the following theorem:

Theorem 2 (Soundness and Completeness of Star Partition)**.** *Star partition is sound and complete.*

Proof. For the soundness, if P is a valid pattern in Sr_s, then for every object $v \neq s, v \in P.O$, and for every timestamp $t \in P.T$, $C_t(v) = C_t(s)$. This means, for every snapshot t, all object in $P.O$ belongs to the same cluster. Thus, P is a valid pattern in original trajectories.

For the completeness, if P is a valid pattern in original trajectories, for an arbitrary object $s \in P.O$, it follows $P.O \equiv S_r$. Since for any time t, $C_s(t) = C_v(t) \forall v \in P.O$, every time in $P.T$ appears in edges in Sr_s, thus P is valid pattern in Sr_s. □

Based on the above theorem, each star can be processed independently by executors, which avoids the inter-executor communication during the mining phase. It is notable that, the replication of data is $O(|\mathbb{O}|^2|\mathbb{T}|)$, which is free from the parameters of patterns. In later sections, we will describe optimization techniques to reduce the replications.

5.2.2 Apriori Mining

For each task, in the mining phase, we need to find the patterns that conform to the parameters. To systematically discover the patterns, we design the *Apriori Mining* method which is similar to frequent item mining. During the algorithm, we call a candidate pattern R-pattern if the size of its object set is $|R|$. Our algorithm runs in iterations. During each iteration R, we try to generate all $(R+1)$-patterns. In iteration 1, the 2-pattern is the edges in Sr_s. In particular, for each $e(s,v) = ET$, pattern $p = (\{s,v\}, ET)$ is formed. During each iteration, we generate $(R+1)$-cluster patterns by joining R-cluster patterns with 2-cluster patterns. Specifically, the join between $p_1 = (O_1, T_1)$ and $p_2 = (O_2, T_2)$ would generate a new pattern $p_3 = (O_1 \cup O_2, T_1 \cap T_2)$. Notice that in Sr_s, each R-pattern consists of the object s, thus the join will grow a R-cluster at most to a $(R+1)$-cluster. Our mining algorithm stops where no further patterns are generated. The algorithm is illustrated as in Algorithm 3.

Algorithm 3 Apriori Mining

```
Require: $Sr_s$
 Lv ← {}
 Ground ← {}
 for all $e(s,t) = T \in Sr_s$ do
     Ground.add($\langle\{s,t\}, T\rangle$);
     Lv ← Ground;
 end for
 while true do
     if Lv is not empty then
         LvCand ← {}
         for all $cand_v \in Lv$ do
             for all $cand$ ∈Ground do
                 $p \leftarrow cand_v$ join $cand$
                 if $p.T$ is partly valid then
                     LvCand.add($p$)
                 end if
             end for
         end for
         Lv ← LvCand
     else
         break
     end if
 end while
```

Algorithm 3 takes exponential complexity to mine GCMP. There are two major factors dragging the performance. First, the size of Sr_s affects the initial size of 2-patterns. Second, the candidates generated in each level affects the join performance. In later sections, we exploit the property of GCMP to reduce the two factors.

6. OPTIMIZATION

We have analyzed that the bottlenecks of Algorithm 3 lies in two factors. The size of each Sr_s and the size of candidates in each level of Apriori. In this section, we provide several optimizations to boost the bottlenecks.

6.1 Edge Reduction by Direction

The first spot for reducing the size of Sr_s is to remove the replicated edges. As shown in Algorithm 2, each edge in the conceptual graph is replicated twice in generating the star-structure. The purpose of replication is to ensure the completeness of star partition. However, this replication can be avoided if we choose an appropriate way of partitioning.

We design the edge partitioning method by edge direction. Instead of building a conceptual graph that is undirected, we create the directed conceptual graph as follows: First, we assign each object a unique number. Then for a cluster C_t in snapshot S_t, for any pair $(u,v) \in C_t$, an edge $e(u,v) = \{t\}$ is created if $u < v$. It is easy to see that the directed conceptual graph is a DAG. We then create each Sr_u by including all the outgoing edges of u. By so doing, each edge is assigned to only one star, thus avoids the replications. We use the following theorem to ensure the completeness of the edge direction method.

Theorem 3 (Sound and Completeness of Edge Direction). *Star partition with edge direction is sound and complete.*

Proof. It is notable that each star is a subset of original trajectories, thus the soundness is trivially true. For completeness, if P is valid pattern, then let s be the object of the smallest number in $P.O$, i.e., $s = \min_{o \in P.O}(o)$. Since s is smallest and the all other objects in $P.O$ is connected with s. Therefore, $P.O \equiv Sr_s$, which indicates that $P.O$ is also a pattern in Sr_s. □

An example of edge direction is shown in Figure 4. As shown, by adapting the direction method, half the size of S_r's is reduced. This clearly brings efficiency in both shuffling and apriori mining.

6.2 Edge Simplification

Each edge $e(s,t)$ in Sr_s contains a time sequence ET which represent the co-occurrence of s and t. We notice that the edge between s and t is not always necessary. For example, if an edge has a cardinality less than K, it is unnecessary to include this edge to Sr_s since it cannot contribute to any patterns. This motivates us to simplify the edges in Sr_s to boost the overall performance.

We first define the *Pseudo-consecutiveness* of a time sequence as follows:

Definition 6 (Pseudo-consecutiveness). *Given a parameter G, a sequence T is* pseudo-consecutive *if and only if for any $i \in [1, |T|]$, $T[i] - T[i-1] \le G$*

For example, let $G = 2$, then the sequence $T_1 = \{1,2,4,5\}$ is pseudo-consecutive while $T_2 = \{1,4,5,6,8\}$ is not pseudo-consecutive.

A pseudo-consecutive subsequence of T is a subsequence $T' \subseteq T$ and T' is pseudo-consecutive. A maximal pseudo-consecutive subsequence T^m of T is a pseudo-consecutive subsequence of T and there is no superset of T^m that is also a pseudo-consecutive subsequence of T. Clearly, a sequence T can be decomposed into several maximally pseudo-consecutive subsequences. We then define a *partly candidate* sequence as follows:

Definition 7 (Partly Candidate Sequence). *Given the pattern parameters: L, K, G, a sequence T is a* partly candidate *sequence if exists one of its maximal pseudo-consecutive subsequence T' such that T' confirms to L, K, G.*

For example, let $L = 2, K = 4, G = 2$, sequence $T_1 = (1,2,4,5,6,9,10,11)$ is a *partly candidate sequence* since $T_1[1:5] = (1,2,4,5,6)$ is a valid pattern wrt. L, K, G. In contrast, $T_2 = (1,2,5,6,7)$ is not a valid partly candidate sequence.

Observing that only partly candidate sequence can be potentially contribute to a pattern. Therefore, given an edge $e(s,t) = T \in Sr_s$, if T is not a partly candidate sequence, it can be pruned from Sr_s. To efficiently test whether a given sequence is partly candidate, we define the *Fully Candidate Sequence*:

Definition 8 (Fully Candidate Sequence)**.** *Given the pattern parameters: L, K, G, a sequence T is a* Fully Candidate *if and only if for any of its maximal pseudo-consecutive sequence T', T' is partly consecutive.*

For example, let $L = 2, K = 4, G = 2$, sequence $T_1 = (1,2,4,5,6,9,10,11)$ is not a fully candidate sequence since one of its maximal pseudo-consecutive sequence $(9,10,11)$ is not a partly candidate sequence. In contrast, sequence $T_2 = (1,2,4,5,6)$ is a fully candidate sequence.

Based on the fully candidate sequence, we can reduce an sequence T to a fully candidate sequence by striping out its non-partly candidate maximal pseudo-consecutive sequences. The reduction works as in Algorithm 4. It takes two rounds of scan of an input T. In the first round of scan, the consecutive portion of T with size less than L is removed. In the second round of scan, the pseudo-consecutive portion of T with size less than K is removed. Clearly the simplification algorithm runs in $O(|T|)$ time.

Algorithm 4 Edge Simplification

```
Require: $T$
  Remove the consecutive portion with size less than $L$
  $c \leftarrow 0$
  for $i \in (0, ..., |T|)$ do
    if $T[i] - T[i-1]! = 1$ then
      if $i - c < L$ then
        $T$ remove $[c : i)$
      end if
      $c \leftarrow i$
    end if
  end for
  Remove the pseduo-consecutive portion with size less than $K$
  $s \leftarrow 1$, $c \leftarrow 0$
  for $i \in (0 : |T|)$ do
    if $T[i] - T[i-1] > G$ then
      if $s < K$ then
        $T$ remove $[c : i)$
      end if
      $c \leftarrow i$
      $s \leftarrow 1$
    else
      $s{+}{+}$
    end if
  end for
```

We use the following theorem to state the completeness and correctness of our edge reduction algorithm.

Theorem 4 (Soundness and Completeness Edge Simplification)**.** *Star partition with edge simplification is sound and complete.*

Proof. Soundness of the star partition is not affected by edge simplification since each star is a subset of original trajectory. For completeness, consider a sequence T, let T_1 be the result of T after first round of simplification. By the nature of the first round of simplification, if $P.T \subseteq T$, then $P.T \subseteq T_1$. In the second round of simplification, only the pseudoconsecutive parts with size less than K are removed. This means, the removed parts is not able to contribute to any patterns. Thus if $P.T \subseteq T$, then $P.T \subseteq T'$. ☐

By leveraging the edge simplification, the size of Sr_s can be greatly reduced. If an edge cannot be reduced to a full candidate sequence, then it is directly removed from Sr_s. If an edge can be reduced to a full candidate sequence, replacing itself by the full candidate sequence results in a more compact storage.

6.3 Pruning Apriori Mining

During the apriori phase, we repeatedly join candidate patterns in different levels to generate a larger set of a patterns. We notice that, such joins can be early terminated by utilizing the monotonic property of GCMP. The intuition is that when we are trying to store a candidate, if its temporal sequence is not a *partly candidate* sequence, then it cannot generate full candidate and thus it can be pruned. This *monotonic property* is explicitly described as in the follow theorem:

Theorem 5 (Monotonic Property of GCMP)**.** *Given the temporal parameters L, G, K, for a candidate cand in Algorithm 3, if cand.T cannot be reduced to a partly candidate sequence, then cand can be pruned.*

Proof. Let $cand_i$ be a i-pattern. If $cand_i$ is generated from $cand_j$ where $j < i$, then if $cand_j.T \subseteq cand_i.T$. That is as the level goes up, the time sequence set of a candidate shrinks. Therefore, if $cand_j.T$ is not a partly candidate sequence, $cand_i.T$ cannot be a partly candidate sequence. ☐

With the help of the *Monotonic Property*, the number of new candidates in each level is greatly reduced. We verify this in the experiment session as well.

7. REFERENCES

[1] X. Li, *Managing moving objects and their trajectories.* PhD thesis, National University of Singapore, 2013.

[2] Y. Li, J. Bailey, and L. Kulik, "Efficient mining of platoon patterns in trajectory databases," *Data & Knowledge Engineering*, 2015.

[3] P. Kalnis, N. Mamoulis, and S. Bakiras, "On discovering moving clusters in spatio-temporal data," in *Advances in spatial and temporal databases*, pp. 364–381, Springer, 2005.

[4] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1099–1108, ACM, 2010.

[5] K. Zheng, Y. Zheng, N. J. Yuan, and S. Shang, "On discovery of gathering patterns from trajectories," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pp. 242–253, IEEE, 2013.

[6] R. Jinno, K. Seki, and K. Uehara, "Parallel distributed trajectory pattern mining using mapreduce," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pp. 269–273, IEEE, 2012.

[7] X. Li, V. Ceikute, C. S. Jensen, and K.-L. Tan, "Effective online group discovery in trajectory databases," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, no. 12, pp. 2752–2766, 2013.

[8] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 330–339, ACM, 2007.

[9] J. Gudmundsson, M. van Kreveld, and B. Speckmann, "Efficient detection of motion patterns in spatio-temporal data sets," in *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pp. 250–257, ACM, 2004.

[10] Y. Wang, E.-P. Lim, and S.-Y. Hwang, "Efficient mining of group patterns from user movement data," *Data & Knowledge Engineering*, vol. 57, no. 3, pp. 240–282, 2006.

[11] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1068–1080, 2008.

[12] Z. Li, B. Ding, J. Han, and R. Kays, "Swarm: Mining relaxed temporal moving object clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 723–734, 2010.

[13] J. Gudmundsson and M. van Kreveld, "Computing longest duration flocks in trajectory data," in *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pp. 35–42, ACM, 2006.

[14] M. R. Vieira, P. Bakalov, and V. J. Tsotras, "On-line discovery of flock patterns in spatio-temporal data," in *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pp. 286–295, ACM, 2009.

[15] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle, *Reporting flock patterns.* Springer, 2006.

[16] M. Andersson, J. Gudmundsson, P. Laube, and T. Wolle, "Reporting leadership patterns among trajectories," in *Proceedings of the 2007 ACM symposium on Applied computing*, pp. 3–7, ACM, 2007.

[17] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[18] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[19] L.-A. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C.-C. Hung, and W.-C. Peng, "On discovery of traveling companions from streaming trajectories," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pp. 186–197, IEEE, 2012.

[20] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan, "Mr-dbscan: An efficient parallel density-based clustering algorithm using mapreduce," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pp. 473–480, IEEE, 2011.

[21] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 593–604, ACM, 2007.

[22] Y. Li, J. Han, and J. Yang, "Clustering moving objects," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 617–622, ACM, 2004.

[23] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F. Özcan, "Clash of the titans: Mapreduce vs. spark for large scale data analytics," *Proceedings of the VLDB Endowment*, vol. 8, no. 13, pp. 2110–2121, 2015.